Title: Measurement and Characterization of Haswell Power and Energy Consumption

Author(s): Huang, Song
Lang, Michael Kenneth
Pakin, Scott D.
Fu, Song

Intended for: 3rd International Workshop on Energy Efficient Supercomputing (E2SC), 2015-11-15 (Austin, Texas, United States)

Issued: 2016-04-04 (rev.1)

Measurement and Characterization of Haswell Power and Energy Consumption

Song Huang[1,2], Michael Lang[2], Scott Pakin[3], and Song Fu[1]
[1]Department of Computer Science and Engineering, University of North Texas
[2]Ultrascale Systems Research Center, Los Alamos National Laboratory
[3]Applied Computer Science Group, Los Alamos National Laboratory
songhuang@my.unt.edu, mlang@lanl.gov, pakin@lanl.gov, song.fu@unt.edu

ABSTRACT

The recently introduced Intel Haswell processors implement major changes relative to their predecessors with respect to power management. Haswell processors are used in NNSA's latest supercomputer, Trinity, hosted at Los Alamos National Laboratory. In this paper we measure and analyze a number of power-based parameters of Haswell that are of great importance for the energy consumption of applications. We study three HPC benchmarks, HPL, STREAM, FIRESTARTER and a hydrodynamics application, CLAMR. These are representative of workloads stressing different components of computers. Our experimental results show that real-time on-board power monitoring consumes substantial power if no optimization is performed; adapting P-states provides a cost-effective way to improve the power-performance of applications; energy savings for hyperthreading was dependent on the application, specifically with the CLAMR application we see 19.8% energy savings with a high thread count, and a 19.5% energy loss for a fewer threads; and HPC applications should employ differentiated core-affinity strategies in order to maximize the performance:power ratio. Moreover, we study the imbalance of sockets on a server in their power and energy use and propose approaches to mitigate such imbalance.

Keywords

Power and energy consumption, high performance computing, Intel Haswell; benchmarking; modeling

1. INTRODUCTION

Power consumption and energy efficiency have become important issues for researchers and engineers in the high-performance computing (HPC) community [15] as the power demand of large-scale HPC systems can easily exceed the power supply of an entire city [1]. Efficient allocation and effective utilization of power and energy in machine rooms are demanded by industry and government sectors.

E2SC2015, November 15-20, 2015, Austin, TX, USA
ACM 978-1-4503-3994-0/15/11 ...$15.00.
DOI: http://dx.doi.org/10.1145/2834800.2834807

HPC systems comprise compute nodes, interconnects, and storage. Of these, the nodes—and within these, the CPUs—typically consume the bulk of the overall power. Consequently, CPU vendors have been incorporating increasingly sophisticated power-saving features in their products. The Xeon E5-2600 v3 family, code-named Haswell-EP, is Intel's latest high-performance processor. It includes advanced power control in addition to other enhancements such as more cores, more memory, and more bandwidth. Most importantly, unlike in previous processors, the voltage and frequency of individual Haswell cores can be adjusted dynamically, at least in the server-class parts [7].

An important question that we address is, *How will new power management techniques influence the power-performance of HPC applications?* In order to make the best use of the power management features on Haswell for energy-efficient high performance computing, we need to measure and characterize them in running HPC applications and benchmarks.

In this paper we evaluate on computer servers equipped with Haswell-EP processors three HPC benchmark codes—HPL, FIRESTARTER, and STREAM—and a proxy application, CLAMR. These codes represent a gamut of HPC applications from completely compute-bound to memory-bound and mixed workloads. We investigate various power management techniques—RAPL power monitoring, P-states, hyperthreading, and core affinity—and their effects on the performance:power ratio of these codes. Our experimental results reveal that without optimization, the on-board power monitoring via RAPL can consume 28.6% more power than idle power. P-states affect the package power rather than the DRAM power, and P-states provide an effective way for energy saving. The benefit of hyperthreading depends on the workload type. Enabling hyperthreading can reduce the execution time and save energy for compute-bound applications. Its effect on memory-bound codes is almost neutral. To maximize the power-performance gain, differentiated core affinity strategies should be employed according to the workload type, which can improve the performance and energy efficiency by 19–48%. Moreover, we observe different patterns of power and energy imbalance between sockets on a server. Factors are identified and approaches are proposed to mitigate the imbalance. Our experimental results and the corresponding findings provide valuable insights into the performance, power and energy of HPC applications run on the new Haswell-EP processors. They guide the development of new, efficient and effective power-management schemes.

The rest of this paper is organized as follows. We first present the Haswell processors' new power-management features in Section 2. The experimental setup including the hardware configuration, HPC workloads, and power-performance metrics and tools, is described in Section 3. Section 4 presents and discusses the experimental results of power and energy consumption on a Haswell-EP server. Future work is discussed in Section 5. Related research is presented in Section 6. Finally, Section 7 draws some conclusions from our measurements.

2. POWER MANAGEMENT IN HASWELL

Haswell includes a number of new power-management features. The two that we leveraged to complete our experiments are *per-core power management* and *independent un-core frequency scaling.*

Per-core power management.

This the main innovation of the Haswell micro-architecture. It provides independent control of each CPU's power consumption. This control includes frequency scaling (DVFS) and power capping. With this functionality we can put some cores in lower power states to enable turbo mode on other cores. This gives us the ability to tune the set of cores for various computational tasks. Note that in our particular experimental setup we were unable to enable per-core power capping due to hardware limitations.

Independent un-core frequency scaling.

On the Haswell there are seperate power control units for each core and the un-core (memory controller, L3 caches, and IO) that are independent from the CPU power plane. In particular, this decouples the CPU frequency from the memory frequency. Hence, one could run at a high rate while the other runs at an reduced rate to better balance application requirements.

Other innovations in power savings.

Some other Haswell power-saving features, which we did not evaluate, include new deeper C-states, new "S0ix" low-power states, more energy-efficient turbo mode, and more aggressive gating of logic. Additionally, Haswell provides seperate AVX base and turbo states. When the processor is using AVX instructions the maximum turbo frequency is lower than without AVX instructions [8].

3. EXPERIMENTAL SETUP

3.1 Hardware

For our experimental evaluation we use two Dell PowerEdge R730 rack servers, each with two Intel Xeon E5-2660 v3 processors, 128GB RAM and one 200GB SSD, which is a typical setting for HPC machines, at least at LANL. Node details are presented in Table 1. We use a Watts Up Pro digital power meter for external power measurements. This power meter provides AC power consumption data for a full node with up to 1 second resolution.

Table 1: Technical specifications of the experimental environment

Compute server	Dell PowerEdge R730
Processor	2x Intel Xeon E5-2660 v3 (Haswell-EP)
Cores/socket	10
Threads/socket	20
CPU frequency	1.2–2.6 GHz
Turbo frequency	3.3 GHz
Cache	25 MB Intel Smart Cache
TDP/socket	105W
Enabled features	Uncore frequency scaling, per-core P-states, energy-efficient turbo

3.2 Workloads

We select four programs as benchmarks to run on the Haswell-based servers and measure the power and energy consumption with various power management approaches applied. These programs represent a gamut of application characteristics from completely compute-bound (HPL and FIRESTARTER) to memory-bound (STREAM) and mixed (CLAMR).

CLAMR (Compute Language Adaptive Mesh Refinement) [14] is a DOE proxy application developed at LANL as a test bed for algorithm development for heterogeneous exascale supercomputers. CLAMR is a cell-based adaptive mesh refinement (AMR) hydrodynamic application that solves the shallow-water equations. The computational model uses Eulerian equations to simulate the fluid flow. We use CLAMR's OpenMP version in our experiments.

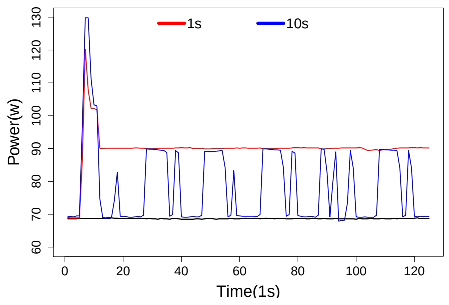
FIRESTARTER [6] aims at creating near-peak power consumption on standard compute nodes. It stresses the most important power consumers: CPU (cores + uncore components) and main memory. It can reliably exceed the power consumption of other stress tests and create steady power-consumption patterns. It can also be used as a maximum power consumption baseline for application energy-efficiency studies.

HPL (High Performance Linpack) [9] solves a (random) dense linear system in double-precision arithmetic on distributed-memory computers. It is a highly scalable MPI program, heavily optimized to squeeze the utmost performance out of parallel machines. With its compute-heavy kernel, HPL approaches the theoretical peak performance (in Gflops/s) of most machines.

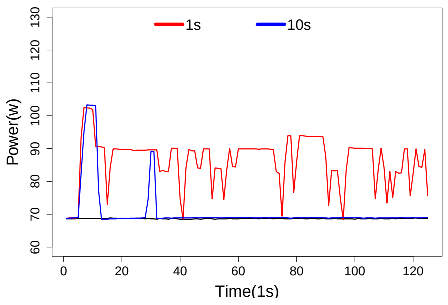
STREAM [13] is a synthetic benchmark designed to measure the sustainable memory bandwidth and the corresponding computation rate of four vector kernels, which perform vector operations on long vectors. STREAM is designed to work with datasets that are much larger than the available cache on a system. The results are intended to be indicative of the performance of very large, vector-style applications. It measures the performance of four long vector operations: "Copy" measures transfer rates in the absence of arithmetic; "Scale" additionally includes a simple arithmetic operation; "Sum" further includes a third operand to enable multiple load/store ports on vector machines to be tested; and "Triad" performs chained/overlapped/fused multiply/add operations.

3.3 Tools and Metrics

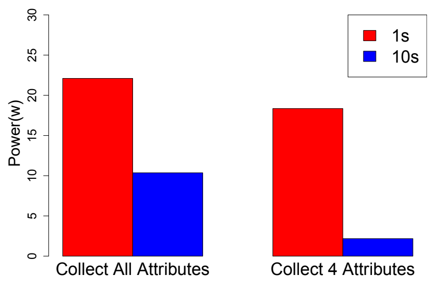
In addition to using an external power meter we also poll for power measurements inband via the Intel Running Average Limit (RAPL) interface. RAPL returns samples averaged over time of various power planes (per core, per package, uncore, and DRAM). RAPL data is made available through machine-specific registers (MSRs). On our particular platform, hardware limitations restricted us to per-package readings and DRAM readings.



(a) Retrieving all 28 RAPL attributes



(b) Retrieving the selected four attributes



(c) Average overhead

Figure 1: Overhead of Power Monitoring

PAPI (Performance API) [20] is a platform-independent library for gathering performance-related data from hardware performance counters. The PAPI interface provides power and energy measurements from RAPL in-line to running programs. It uses the MSR driver to gather RAPL values.

PAPI and the external power meters report power measurements periodically (at possibly different frequencies). If the power consumption of an HPC workload is steady, we can take the mean power consumption P and multiply it by the execution time T of the workload, which results in the energy consumption E in joules: $E = P \cdot T$. We can use this metric when the resource usage is steady.

4. EXPERIMENTAL RESULTS

In this section, we present measured power and energy consumption while running HPC-style workloads (Section 3.2) on our Haswell-EP compute nodes. We are particularly interested in evaluating the overhead of power monitoring and the influences of hyperthreading, core affinity, and various P-states on power consumption and performance.

4.1 Overhead of Power Monitoring

RAPL enables on-board power monitoring of the processor and DRAM without using external power meters. By reading the RAPL MSRs, PAPI can record real-time data of the power and energy consumption. The thread that runs PAPI (we call it the *PAPI thread*) is executed on a CPU core and the retrieved MSR readings are stored in DRAM, both of which draw power. In the first set of experiments, we aim to analyze this overhead. To this end, we use the external power meter to measure the node-wide power consumption both in the idle state and during the execution of the PAPI thread. In the idle state, only minimal OS activity (e.g., interrupt handlers and a few core dæmons) is present, and this consumes very little power.

Figure 1 shows the extra power drawn for on-board power monitoring. The idle power of the server is 69W. Because the resolution of the Watts Up Pro digital power meter is no finer than one second, we measure the monitoring overhead as the sampling rate of the PAPI thread at 1-second and 10-second intervals. By default, the PAPI thread retrieves all 28 RAPL attributes. Figure 1a shows that the overhead of one-second monitoring is steady while that of 10-second monitoring fluctuates. Moreover, from Figure 1c we can see the overhead of one-second monitoring is more than twice that of 10-second monitoring which is 10.4W on average.

However, not all of the 28 RAPL attributes are relevant to power and energy monitoring. Some of them are not even supported by the E5-2660 v3 processor, such as those for the PP0 and PP1 power planes. We select four RAPL attributes for power monitoring. They are Package_ENERGY:Package0, DRAM_ENERGY:Package0, Package_ENERGY:Package1, and DRAM_ENERGY:Package1. Figure 1b shows that the power draws from both one-second and 10-second monitoring drop and the latter becomes more sporadic. The average power draw in Figure 1c indicates that using four RAPL attributes can reduce the overhead by 78.9% (10-second rate) and 16.9% (1-second rate) relative to using all 28 attributes.

On-board RAPL power monitoring by PAPI can cause up to 32% more power draw than the idle power in the steady state. This overhead can be reduced by using longer monitoring period and less RAPL attributes. Combining this two achieves a 90.1% decrease of the overhead in our experiments. In production runs, other factors, such as the granularity of power control and complexity of power management mechanism, will also affect the selection of power monitoring parameters.

Moreover, we measured the power usage of other components, such as disk and network, when running the benchmarks. The average power consumption is 18W.

4.2 Effects of P-States on Applications' Power-Performance

As listed in Table 1, the E5-2660 v3 Haswell-EP processor supports a frequency range from 1.2 GHz to 2.6 GHz and a turbo frequency at 3.3 GHz. In our next set of experiments, we vary P-states and measure the performance and power consumption of running the four benchmark programs. We scale the P-states to 1.2, 1.3, 1.5, 1.7, 1.9, 2.1, 2.3, and 2.5 GHz with the "powersave" governor and to 2.9 GHz with the "performance" governor.

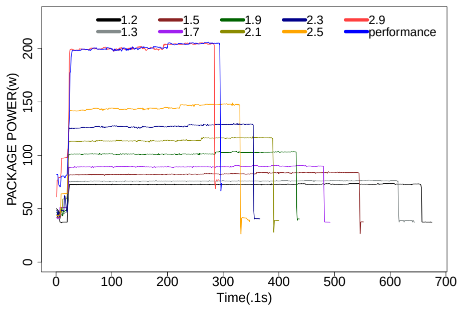
Figure 2 shows the package power draws by HPL, STREAM, and CLAMR and Figure 4a is for FIRESTARTER. In the figures, we can see the changing of P-states has a direct and clear effect on the power use of all four applications. The higher the P-state is, the more power is drawn until it hits the turbo frequency, that is 2.8-3.3 GHz. Related research [19] shows that the power use of the CPU follows

$$P = c \cdot V^2 \cdot f + P_s \quad (1)$$

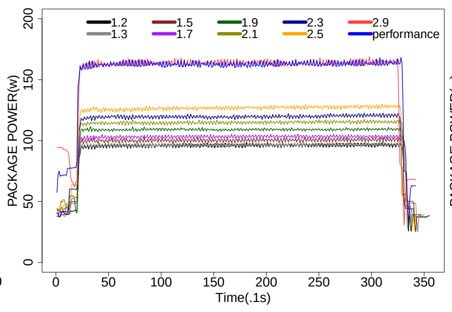
with

$$f \propto (V - V_{th})^\alpha / V \quad (2)$$

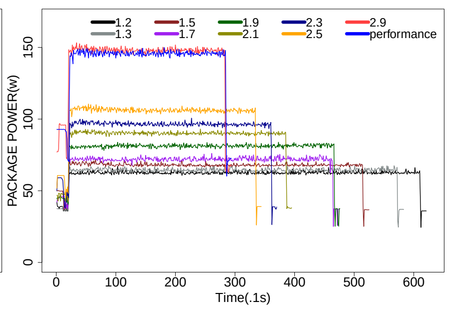
as $V \gg V_{th}$, $P \propto f^{(\alpha+1)/(\alpha-1)}$. According to our experimental results, the value of α for the Haswell-EP processor is between 1.22 and 1.49.



(a) HPL (compute bound)

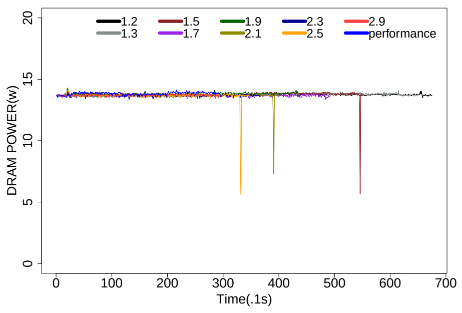


(b) STREAM (memory bound)

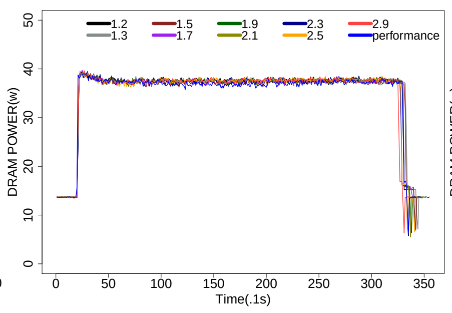


(c) CLAMR (mixed)

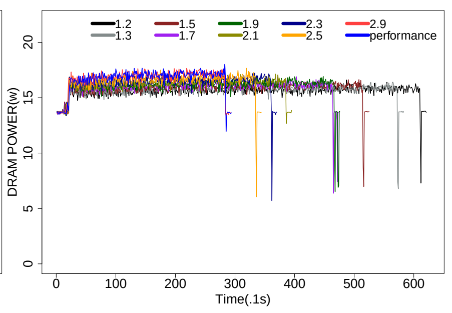
Figure 2: Package power draw from running benchmark applications at different P-states



(a) HPL (compute bound)

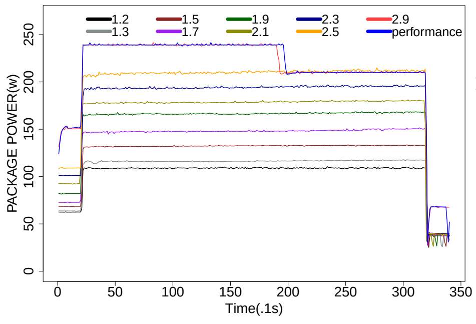


(b) STREAM (memory bound)

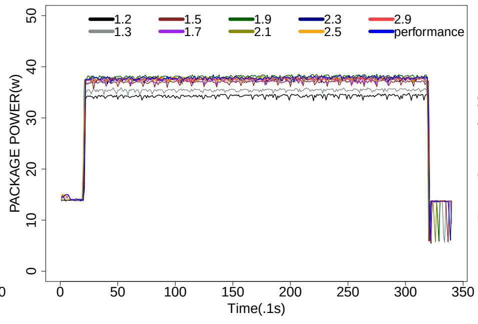


(c) CLAMR (mixed)

Figure 3: DRAM power draw from running benchmark applications at different P-states.



(a) Package power draw



(b) DRAM power draw

Figure 4: Power consumption of FIRESTARTER with different P-states

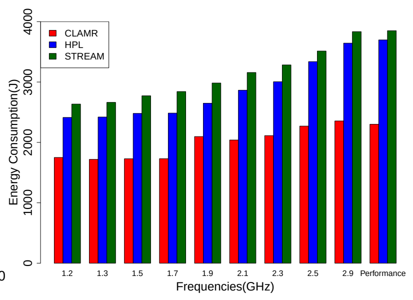
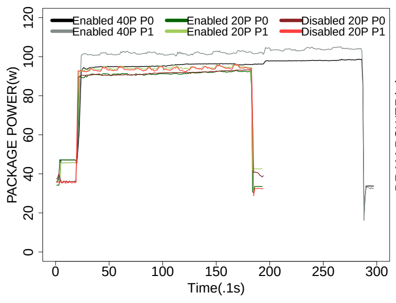
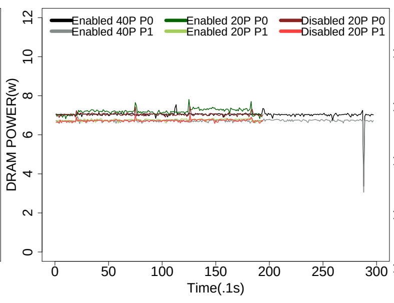


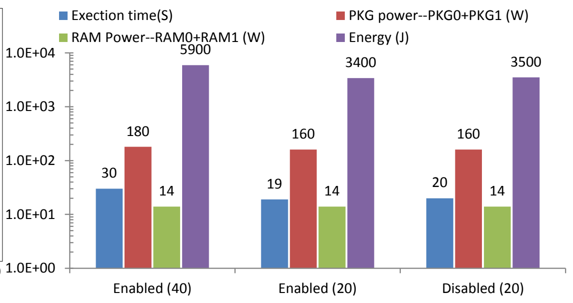
Figure 5: Energy consumption with P-states



(a) Package power use



(b) DRAM power use



(c) Comparison

Figure 6: Power-performance of running HPL (compute bound) with and without hyperthreading

The influence of P-states on the power-performance of different types of applications is different. For compute-bound HPL, increasing the P-state from 1.2 GHz to 2.6 GHz leads to a 50.7% decrease of the execution time. An additional 8.2% decrease is achieved by using the turbo frequency. A similar trend is observed when running CLAMR. However, for memory-bound STREAM, the change of P-states does not yield considerable benefit for the performance, as the execution time is determined by the memory bandwidth instead of CPU's P-states. In contrast to package, the power consumption of DRAM is insensitive to P-states.

Another interesting finding is that we did not observe clear turbo boost during the execution of HPL, STREAM, and CLAMR, as the power draw of each package is no higher than 105W, that is the TDP. In contrast, FIRSTARTER can achieve the near-peak power use. In Figure 4, we observe that the package power consumption reaches 120.8W for 17.5 seconds before an abrupt drop to the TDP, as the thermal envelop is used up, which terminates the turbo boost. Unlike the other three codes whose workloads are determined by their problem sizes, FIRESTARTER is a test code that continues running until it is manually terminated. Therefore, its energy use data is not comparable with those of the other applications.

Figure 5 presents the energy consumption of HPL, STREAM, and CLAMR run at different P-states. Generally speaking, the higher energy efficiency is achieved at the lower P-states. For example, HPL, STREAM, and CLAMR consume the least amount of energy when the P-state is at 1.2, 1.2, and 1.3 GHz, respectively. The highest energy uses are all achieved at the turbo frequency. A reason for this phenomenon is that we conduct our experiments on a single node. When running MPI applications on multiple nodes, the inter-node communication and phase synchronization will complicate the energy use pattern. HPL achieves the highest ratio of maximum to minimum energy consumption, which is 1.53 times.

4.3 Effects of Hyperthreading on Power- Performance

Intel Xeon E5-2660 v3 processors support two-way hyperthreading. Because each processor is equipped with 10 cores, in total 40 hyperthreads are available simultaneously on the 20 physical cores. In this set of experiments, we evaluate the impact of hyperthreading on the execution time, power and energy use of the benchmark applications. Because the number of application threads of FIRESTARTER is not configurable, we focus on the other three codes: HPL, STREAM, and CLAMR.

In our experiments, hyperthreading is always enabled in the BIOS. We enable and disable hyperthreading by setting the logical CPU cores "online" and "offline" respectively in the OS. Two levels of workload intensity are tested, that is *high* (40 application threads are run) and *low* (20 application threads are run).

Figures 6a and 6b show the power consumption of the package and DRAM while running HPL with and without hyperthreading and at the two workload levels. We observe the execution time decreased from 29s to 19s (34%) after disabling hyperthreading under the high workload. This is mostly caused by the contention among HPL threads when running on fewer CPU cores. When the low workload is employed, the execution time, package power, and energy use drop as the contention and synchronization overheads reduce. Enabling hyperthreading can slightly (by 2.9%) reduce the energy consumption. By switching from the high workload to the low workload, the energy consumption drops from 5,900J to 3,400J (42.4% decrease) with hyperthreading.

Compared with HPL, the effects of hyperthreading on STREAM (in Figure 7) and CLAMR (in Figure 8) are different. STREAM is memory bound. The contention under the high workload caused by disabling hyperthreading does not significantly affect the execution time, i.e., a 17.7% increase, and 14.2% and 10.5% decreases of package and DRAM power respectively. The overall effect is that the energy use is increased by 1.7%. Moreover, using the low workload does not yield much energy saving. A similar pattern is observed for CLAMR. However, hyperthreading causes CLAMR to save 19.8% energy under the high workload but consume 19.5% more energy under the low workload.

The above results indicate that hyperthreading is preferable in terms of energy savings, especially for compute-bound applications and when the number of application threads is large.

4.4 Power-Energy Imbalance of the Two Sockets

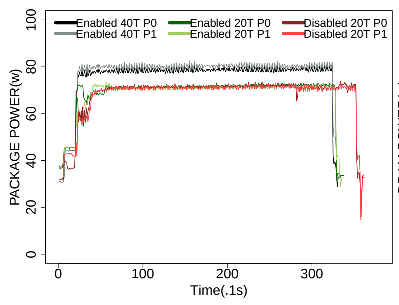
The compute server under test has two sockets, each with a Xeon E5-2660 v3 Haswell-EP processor. In the experiments, we find that the two sockets do not display the same power and energy use. Thus we aim to quantify the extent of such imbalance and find out how to mitigate it.

Figure 9 presents the difference of the power consumption between the two sockets when running HPL, STREAM, and CLAMR at the two workload levels, with and without hyperthreading. Compared with the total package power use of the two sockets in Figures 6-8, the package power imbalance is not significant, ranging from 0.3% to 5.9%. The highest imbalance happens when CLAMR is run at the low workload level with hyperthreading enabled. Socket 0 consumes 7.3 W more power than Socket 1. An explanation is that the 20 CLAMR threads are not evenly distributed on the two sockets (40 hyper threads in total), with more threads being executed on Socket 0 than Socket 1.

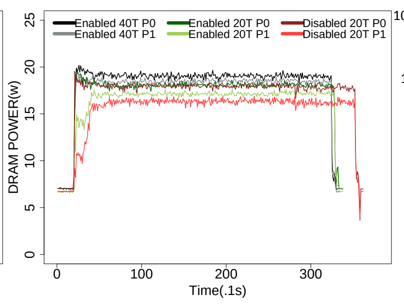
Among the three tested applications, HPL and CLAMR experience more imbalance of the package power than STREAM, as HPL is more compute intensive and thus its package power is more sensitive to the imbalance. Another interesting observation is that for HPL, Socket 1 always draws more power than Socket 0. In contrast, for both STREAM and CLAMR, this bias happens only when the high workload level is employed and hyperthreading is enabled. A possible explanation is that the workloads among HPL threads are not equivalent.

Compared to the package power, the DRAM power, as shown in Figure 9b, has clearer patterns of the imbalance: for STREAM and CLAMR, enabling hyperthreading or halving the high workload can mitigate the power imbalance. However, for HPL, enabling hyperthreading does not help balance the DRAM power. Moreover, the DRAM power has less variability of imbalance than the package power, as the ratio of the highest to the lowest imbalance is 5.7 (DRAM power) versus 18.4 (package power), as the total power consumption of DRAM is much lower than that of package.

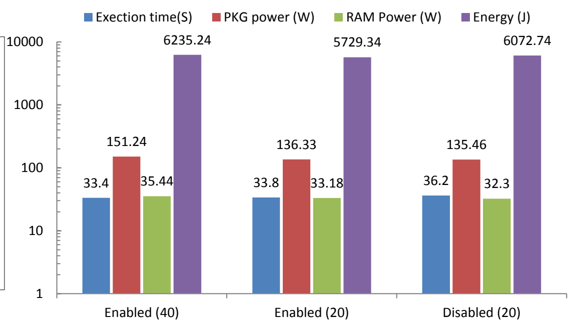
Figure 10 shows the energy imbalance of the two sockets.



(a) Package power use

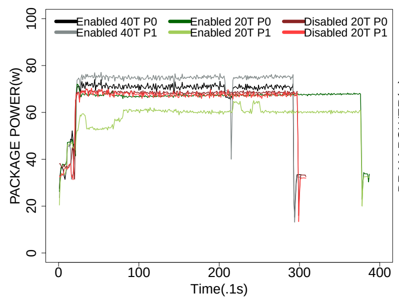


(b) DRAM power use

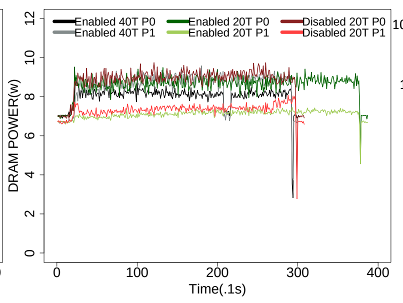


(c) Comparison

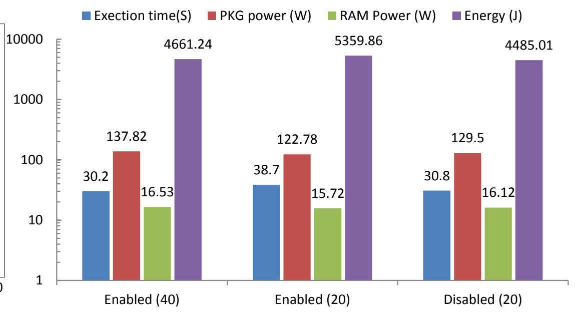
Figure 7: Power-performance of running STREAM (memory bound) with and without hyperthreading



(a) Package power use

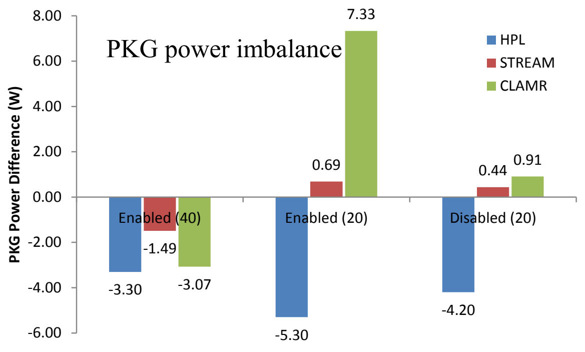


(b) DRAM power use

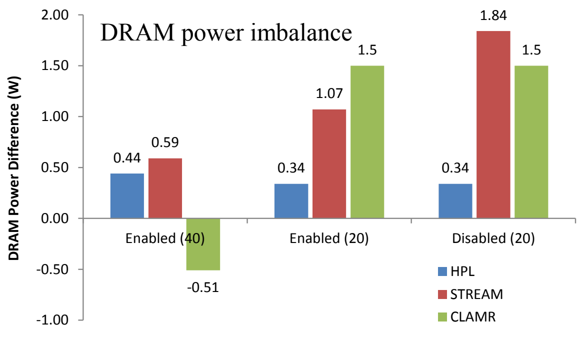


(c) Comparison

Figure 8: Power-performance of running CLAMR (mixed) with and without hyperthreading



(a) Package power imbalance (PKG0 − PKG1)



(b) DRAM power imbalance (RAM0 − RAM1)

Figure 9: Imbalance of the two sockets in power consumption

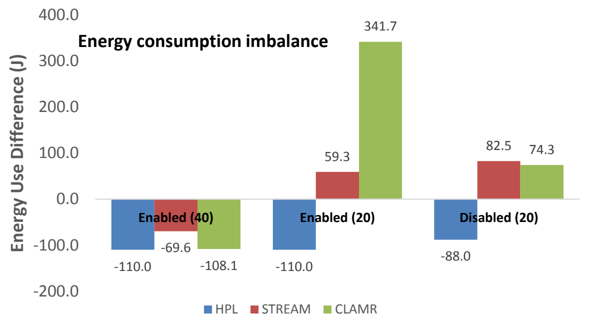


Figure 10: Imbalance in energy consumption

Overall, the imbalance is not very significant, considering the much higher total energy consumption shown in Figures 7–9.

4.5 Effects of Core Affinity on Power-Performance

In all of the preceding experiments, the application threads and PAPI thread can run on any CPU cores (floating). Core affinity (also called thread affinity [16]) enables us to pin an application thread to a specific core. This set of experiments is designed to characterize the influence of core affinity on the power-performance of the benchmark applications run on Haswell-EP processors.

Because we need to deal with both application threads and the PAPI thread, we consider four cases: 1) both types of threads are floating, denoted by $(App_{float}, PAPI_{float})$; 2) the PAPI thread is pinned to a core while letting the application threads float, denoted by $(App_{float}, PAPI_{pinned})$; 3) the two types of threads are exchanged in the preceding case, as $(App_{pinned}, PAPI_{float})$; 4) both types of threads are pinned to specific CPU cores, as $(App_{pinned}, PAPI_{pinned})$. We have already studied Case 1. In this section, we characterize the power-performance of Haswell-EP in the other three cases.

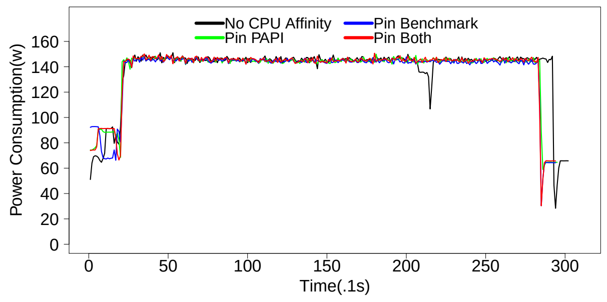


Figure 11: Package power use of CLAMR with core affinity

Figures 11 and 12 show the results of running CLAMR in the four affinity cases. 40 CLAMR threads and one PAPI thread are run with hyperthreading enabled. In the figures, we can see that employing different affinity strategies does not lead to significant difference in the power consumption. However, the execution time is slightly reduced (by 2.6%) by having at least one type of threads pinned compared to Case 1. HPL and STREAM display the similar pattern. We omit their details due to space limit.

Figures 13 and 14 present the analytic results of the effects of core affinity on the execution time, package and DRAM power and energy consumption of the three applications. In

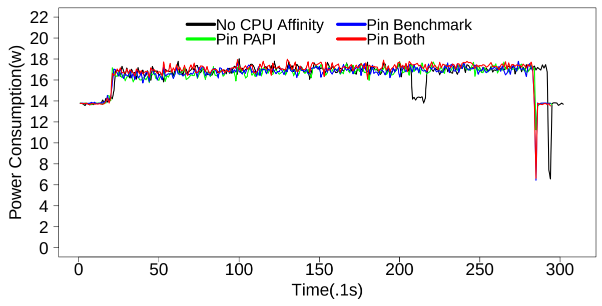


Figure 12: DRAM power use of CLAMR with core affinity

the first set (Figure 13), hyperthreading is enabled, which enables the 40 application threads to run with 40 hyperthreads without contention. In the other set (Figure 14), the 40 threads are run on 20 cores with hyperthreading disabled, which introduces contention. It is clear based on the two figures that core affinity has marginal effects (marginal benefit for CLAMR and marginal negative for HPL and STREAM).

In contrast, when contention exists, as shown in Figure 14, core affinity substantially improves the performance and energy saving. For example, by pinning the PAPI thread to a core (Case 2), we can reduce the execution time of HPL (Figure 14a) by 48% and thus save 48.2% energy, although the package and DRAM powers remain unchanged. Cases 3 and 4 can also reduce 25% and 18.8% energy consumption, respectively. STREAM, however, does not benefit that much from core affinity. At most 8.2% of energy is saved in Case 3 compared to Case 1. Core affinity causes slight increase of the package and DRAM power by 5.3% and 3.8%, respectively. For CLAMR, core affinity saves a substantial amount of energy, i.e., 19.3% in Case 3, 18.6% in Case 4 and 1.9% in Case 2. The execution time is reduced by 23.9%, 23.7%, and 3.1%, respectively.

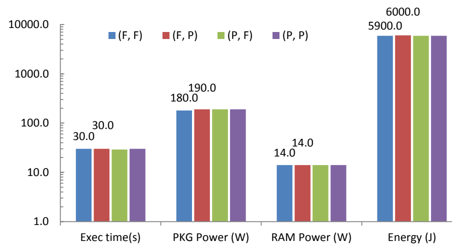
Therefore, when contention exists, core affinity can improve the power-performance of applications. The more compute-intensive an application is, the more benefit core affinity, in particular Case 2 with the pinned PAPI thread, will result in. With the increase of memory-boundness, pinning application threads, i.e., Case 3, will yield better power-performance.

5. DISCUSSION AND FUTURE WORK

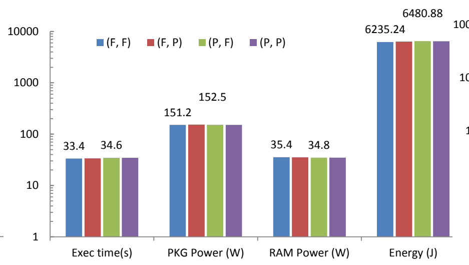
In our experiments, we measure the performance, power and energy consumption of four selected benchmark applications with varying workload patterns run on a computer server equipped with Haswell-EP processors.

An important issue that we have not yet addressed is the inter-node communication and synchronization while running MPI applications on multiple nodes. The performance variability of different nodes will add another dimension of complexity to the power and energy optimization. We are currently expanding the test to run HPC applications in a multi-node environment and analyze the applications' power-performance. To reduce the performance variability among nodes and CPU cores, we will explore the Turbo State Limiting technology introduced by Haswell-EP to put a cap on the turbo states of cores in order to enable applications to run consistently across nodes.

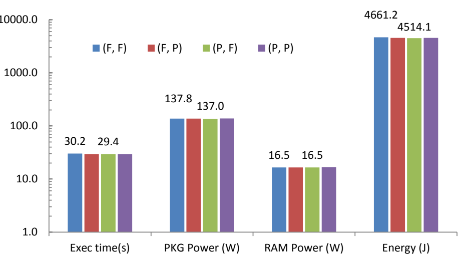
The findings presented in Section 4 are based on the experimental results obtained from running the selected applications. As they represent different types of workload,



(a) HPL

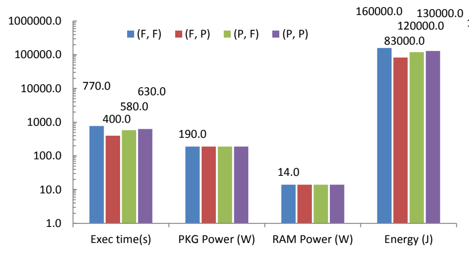


(b) STREAM

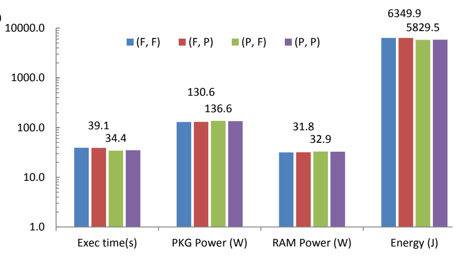


(c) CLAMR

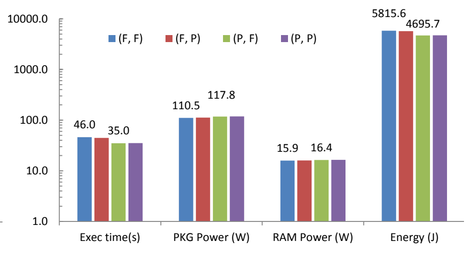
Figure 13: Effects of Core Affinity on Power-Performance without Thread Contention. (App, PAPI) threads are either F (floating) or P (pinned to cores)



(a) HPL



(b) STREAM



(c) CLAMR

Figure 14: Effects of Core Affinity on Power-Performance with Thread Contention

people may challenge the generality of these findings. Our goal is to cover the entire application spectrum, from compute bound to memory bound programs. Our results show that pure compute bound (e.g., HPL) and pure memory bound (e.g., STREAM) have different power-performance characterizations when they are run on Haswell-EP. The one in-between (e.g., CLAMR) has the combined characterizations, which can be viewed as a verification of the findings. To directly validate our findings, we plan to test more HPC applications and use experimental results from applications with similar workload types to cross-validate the findings.

In the preceding experiments, the P-states of all cores in a socket is changed and set to the same frequency level, which is commonly used in today's HPC systems. Haswell also provides a voltage and frequency domain for each core, called Per-Core Power State (PCPS). PCPS enables us to change the cores' frequency independently. Moreover, we can scale the frequency of the uncore, such as LLC and the interconnect rings, up and down independently of the cores, that is Uncore Frequency Scaling (UFS). We will extend our work by designing experiments to test and characterize PCPS and UFS of Haswell-EP from running HPC applications.

6. RELATED WORK

Power profiling in production computer systems provides valuable data and knowledge for developing power simulators and resource scheduling policies. Fine-grained power profiling techniques measure the power usage of individual hardware components, such as CPU [12], memory [22], disk [23] and other devices [21]. In contrast, coarse-grained power profiling aims to characterize system-wide power dynamics, such as the macropower framework [24]. Kamil et al. profiled HPC applications on several test platforms and projected the power profiling results from a single node to a full system [10]. Ge et al. studied the influence of software and hardware configurations on the system-wide power consumption [5]. They found that characteristics of HPC applications affect the power usage of a system. Hackenberg et al. performed a detailed analysis of Haswell's P-state and C-state transition latencies and the impact of Haswell's new power-management mechanisms on memory bandwidth and performance reproducibility [7]. Our paper distinguishes itself from these earlier efforts by measuring and analyzing the impact of Haswell's new power management features on the power and performance of HPC-style codes.

To control the power usage of HPC systems, power capping [2] is a promising and effective approach. System operators can balance the performance and power consumption of clusters by adjusting the maximum amount of power (a.k.a. power budget) that can be consumed by clusters. Pelly et al. presented a dynamic power provisioning and capping method at the [18] power distribution unit (PDU). They proposed to shift the slack power capacity to servers with growing power demand by using a heuristics policy. For HPC jobs, many factors affect the power usage including hardware configurations and resource utilization. Femal et al. developed a hierarchical management policy to distribute power budget among clusters [4]. Kim et al. investigated the relation between CPU voltages and system performance and power efficiency [11]. By exploiting the dynamic voltage scaling (DVS) technologies, they proposed a task scheduling policy which aims to minimize the energy consumption while satisfying the specified performance requirements. Rountree et al. proposed policies of overprovisioning hardware with hardware-enforced power bounds and system-wide power reallocation in an application-agnostic manner [3, 17]. We have developed a full system simulator, TracSim [25], which estimates the trapped power capacity under different power capping and job scheduling policies. This related research is complementary to our work in that our findings can contribute to the design of effective job and resource management techniques for energy-efficient high-performance computing.

7. CONCLUSIONS

Power and energy have become increasingly important components of overall system behavior in high-performance computing. In this work we evaluate the power management techniques of the fourth-generation Haswell-EP processors, including RAPL power monitoring, P-states, hyperthreading, core affinity, and their effects on the power-performance of HPC benchmarks.

We found that monitoring power onboard via RAPL consumes 28.6% more power than idle system. Different settings of P-states could provide efficient way to reach 33.3% energy saving. Enabling hyperthreading and core affinity optimizing are also approaches for energy efficiency. Differentiated core affinity strategies could improve performance and energy efficiency by 19–48%. At the same time, power and energy imbalance can be observed between the two sockets. Our experimental results and findings provide valuable information for the development of new, efficient and effective power management schemes.

8. ACKNOWLEDGEMENTS

This work was performed at the Ultrascale Systems Research Center (USRC) at Los Alamos National Laboratory, supported by the U.S. Department of Energy contract DE-AC52-06NA25396. The publication has been assigned the LANL identifier LA-UR-15-26774.

9. REFERENCES

[1] R. Brown et al. Report to Congress on server and data center energy efficiency: Public law 109-431. *Lawrence Berkeley National Laboratory*, 2008.

[2] J. Choi, S. Govindan, B. Urgaonkar, and A. Sivasubramaniam. Profiling, prediction, and capping of power consumption in consolidated environments. In *Proc. of MASCOTS*, 2008.

[3] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz. POW: System-wide dynamic reallocation of limited power in HPC. In *Proc. of HPDC*, 2015.

[4] M. E. Femal and V. W. Freeh. Boosting data center performance through non-uniform power allocation. In *Proc. of IEEE ICAC*, 2005.

[5] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671, 2010.

[6] D. Hackenberg, R. Oldenburg, D. Molka, and R. Schone. Introducing FIRESTARTER: A processor stress test utility. In *Proc. of IGCC*, 2013.

[7] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer. An energy efficiency feature survey of the Intel Haswell processor. *Proc. of IEEE IPDPSW*, 2015.

[8] P. Hammarlund, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, et al. Haswell: The fourth-generation Intel core processor. *IEEE Micro*, (2):6–20, 2014.

[9] HPL. HPL: High Performance Linpack, 2003.

[10] S. Kamil, J. Shalf, and E. Strohmaier. Power efficiency in high performance computing. In *IPDPS*, 2008.

[11] K. H. Kim, R. Buyya, and J. Kim. Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters. In *CCGrid*, 2007.

[12] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, and S. Dropsho. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. *ACM SIGARCH Computer Architecture News*, 31(2):14–27, 2003.

[13] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. 1995.

[14] D. Nicholaeff, N. Davis, D. Trujillo, and R. Robey. Cell-based adaptive mesh refinement implemented with general purpose graphics processing units. Technical report, Los Alamos National Lab, 2012.

[15] S. Pakin, C. Storlie, M. Lang, R. E. Fields, E. E. Romero, C. Idler, S. Michalak, H. Greenberg, J. Loncaric, R. Rheinheimer, et al. Power usage of production supercomputers and production workloads. *Concurrency and Computation: Practice and Experience*, 2013.

[16] S. Parete-Koon. Process/thread affinity, 2015.

[17] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proc. of ICS*, 2013.

[18] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood. Power routing: dynamic power provisioning in the data center. In *ACM Sigplan Notices*, volume 45, pages 231–242, 2010.

[19] C. Piguet. *Low-power electronics design.* CRC press, 2004.

[20] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring energy and power with PAPI. In *ICPPW*, 2012.

[21] T. T. Ye, G. D. Micheli, and L. Benini. Analysis of power consumption on switch fabrics in network routers. In *Proc. of ACM DAC*, 2002.

[22] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *Proc. of ACM DAC*, 2000.

[23] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, et al. Modeling hard-disk power consumption. In *FAST*, 2003.

[24] Z. Zhang and S. Fu. Macropower: A coarse-grain power profiling framework for energy-efficient cloud computing. In *IPCCC*, 2011.

[25] Z. Zhang, M. Lang, S. Pakin, and S. Fu. Trapped capacity: scheduling under a power cap to maximize machine-room throughput. In *Proc. of E2SC*, 2014.